Texas Instruments

Designer's Guide

SLAU543–December 2013

MSP430TCH5E Haptics Library

By adding haptics effects to your application, you can engage more of the user's senses and enrich the user experience. TI provides a complete easy-to-use solution for this, in the form of the MSP430TCH5E Haptics Library. The library is built around Immersion Corporation's TouchSense™ TS2200 Library, and it provides a means of implementing haptics without patent infringement.

The library runs on the MSP430TCH5E device, interfacing with an ERM or LRA haptics actuator through a DRV2603 or DRV8601 haptics driver device. Although this combination could be run as a fixed-function serial slave device, the MSP430TCH5E creates new possibilities by allowing you to add custom functionality to the MSP430TCH5E application code itself, driving its own haptics based on other inputs – for example, from capacitive touch buttons.

This designer's guide describes how to incorporate the library into a project, and it documents both the API and serial interfaces.

The library described in this document is included with the Capacitive Touch With Haptics Software Development Kit (MSP430-HAPTOUCH-SDK).

Contents

List of Figures

1 Introduction

TI's MSP430TCH5E Haptics Library, running on the MSP430TCH5E device, provides an easy way to add haptics effects to any electronic device. Haptics effects enrich any user experience by engaging more of the user's senses.

Haptic feedback is the use of vibration within an electronic product to interact with the user by his or her sense of touch. By far the most common example is the use of vibration in cell phones as a way of notifying the user of a call, avoiding the need to generate sound. But its applications go far beyond this. Too often underutilized in electronic devices, touch is a powerful sense, and if utilized it can create an experience that is more natural and organic to the way humans normally interact with the world.

Perhaps one of the most growing uses of haptics in the last ten years has been providing tactile feedback for capacitive touch controls. Because of the synergy between haptics and capacitive touch, this library is designed for easy application alongside the Capacitive Touch Software Library (CAPSENSELIBRARY). Similarly, the MSP430TCH5E device is designed to run an application that uses both libraries simultaneously. For an example of this, see the application note *Haptics and Capacitive Touch Using the MSP430TCH5E: the HapTouch BoosterPack* (SLAA616).

The MSP430TCH5E Haptics Library is built around the TouchSense TS2200 library from Immersion, the industry leader in haptics technologies. Using the MSP430TCH5E solution allows implementation of haptics without patent infringement and without royalties.

It is important to note that the MSP430TCH5E Haptics Library only runs on the MSP430TCH5E device. If executed on another MSP430™ device, the application will build, but the API functions will return an error. It was also designed specifically for use with two of TI's haptics driver devices: the DRV2603 and the DRV8601. It has been optimized for the DRV2603, but the DRV8601 has also been tested and is recommended as well.

This designer's guide addresses the software necessary to implement haptics using the MSP430TCH5E. It also covers some basic aspects of implementing the hardware.

The software for this library is included as part of the HapTouch BoosterPack SDK (MSP430-HAPTOUCH-SDK).

Features:

- Quickly add haptics to any application
- Generate any one of 123 haptics effects, or build sequences of those effects
- Generate continuous haptics based on an audio signal, Immersion's Audio2Haptics™ feature
- A complete library and several examples
- The HapTouch GUI makes it easy to choose your effect or build effect sequences

2 Haptics System Overview

This section describes, at a high level, how a haptics system commonly works, and specifically how it is implemented with the MSP430TCH5E.

2.1 *Actuator Interface*

All haptics systems use *actuators* to convert electrical signals into motion.

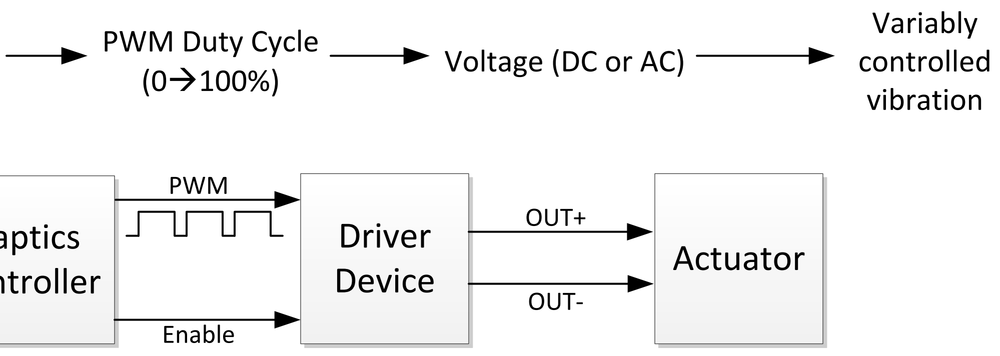


Figure 1. Haptics System

The haptics controller – in the case of this library, the MSP430TCH5E – drives the actuator using a simple pulse-width modulated (PWM) waveform, through a driver device – in the case of this library, the DRV2603 or DRV8601.

In a very generalized sense, there is a direct relationship between the controller's output PWM duty cycle, the voltage produced by the driver device, and the degree of vibration that results from the actuator. That is, the higher the PWM's duty cycle, the more strongly it drives the actuator, and vice versa. Although not entirely this simple, one can think of the PWM as being averaged into a dc voltage, where the higher the voltage, the faster the actuator is.

2.2 *Actuator Types*

There are several actuator types, but two of the most common are:

- ERM: Eccentric Rotating Mass. A motor spins an unbalanced mass. The resulting centripetal force pulls the actuator as it spins; the result is vibration.
- LRA: Linear Resonant Actuator. A coil electromagnetically moves a mass outward along the axis of its center (linearly). This force is counteracted in the opposite direction by a spring, such that when the current through the coil stops, the mass returns to center. As the mass moves, it pulls the actuator body; the result is vibration.

Both types are supported by this solution.

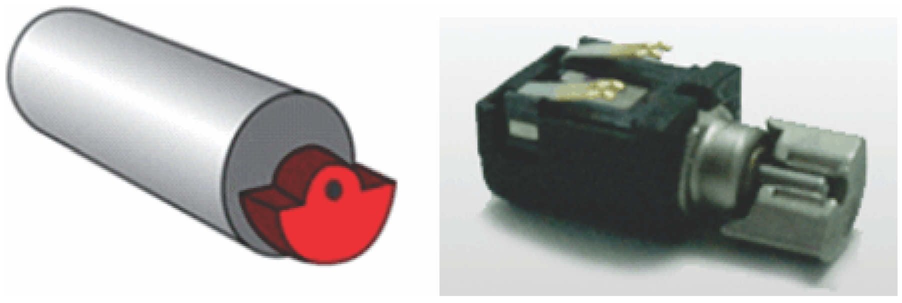


Figure 2. ERM Actuator

In Figure 2, notice that all of the mass is positioned on one side of the axle to maximize the lack of balance.

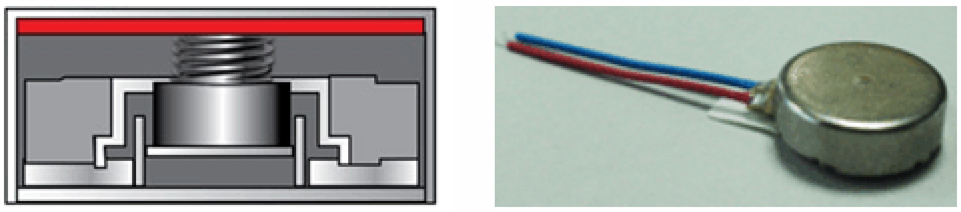


Figure 3. LRA Actuator

LRA actuators often are made in a coin-style form factor (see Figure 3). Although ERM actuators have a longer history, LRA actuators have advantages in power efficiency, lifespan, and response times, and thus are increasing in popularity.

2.3 PWM Frequency

With ERM actuators, the frequency of the PWM waveform usually is not a major concern. It need only be high enough to properly interact with the low-pass filter created by the inductive windings in the motor, toward generating a dc voltage.

LRA actuators are more sensitive to frequency, typically having a window no wider than ±2.5 Hz in which sufficient resonance is achieved.

If using the DRV2603, the frequency of the input PWM is not related to the frequency of the output. The DRV2603 uses auto-resonance in driving the actuator, and the frequency of the input can be anywhere between (at the time of this writing) 10 to 250 kHz, whether driving ERM or LRA. (Always check the driver device data sheet for actual specifications.)

However, if using the DRV8601 with an LRA actuator, it is important to tune the frequency to the chosen LRA actuator.

2.4 Relationships Between Duty Cycle and Vibration

Just as the mechanism of creating motion is fundamentally different between these actuators, so are the waveforms required to drive them. The ERM is driven by a dc voltage, while the LRA is driven by a sinusoidal voltage. (The ERM can actually have the PWM applied to it directly, and the inductance of its windings produces a low-pass filter that averages to a dc voltage.)

Even so, there are things they have in common with respect to the relationship between drive voltage and vibration strength. This is shown in Figure 4.

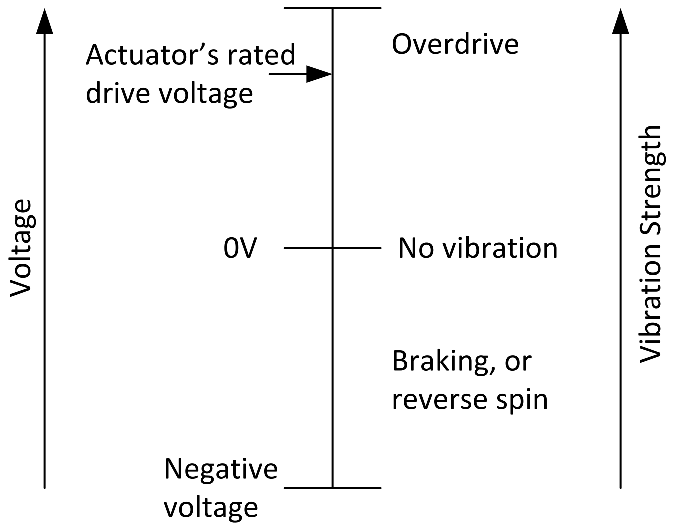


Figure 4. Generalized Relationship Between Actuator Drive Voltage and Vibration Strength

Figure 4 introduces two important terms:

- Overdrive: the temporary application of a voltage beyond the actuator's rated continuous voltage, for the purpose of more quickly moving the mass when it has not been moving.
- Braking: the application of a negative voltage, to more quickly stop the forward motion of the mass.

Although these are handled automatically by the MSP430TCH5E haptics system, they become relevant to the developer when choosing an actuator – see Section 4.2.

In Section 2.1, it was said there is a direct relationship between the controller's output PWM duty cycle, the voltage produced by the driver device, and the degree of vibration that results from the actuator. While true in general, the driver device actually exerts a great deal of influence in how the PWM duty cycle is mapped to vibration. In short, the mapping does not have to be linear.

For example, the DRV2603 and DRV8601 have features that exploit the natures of ERM and LRA actuators toward the goals of maximizing haptic performance and power efficiency. They handle overdrive and braking automatically, such that the controller does not need to consciously implement them. In so doing, they present to the controller a simple relationship between PWM duty cycle and vibration (see Figure 5).

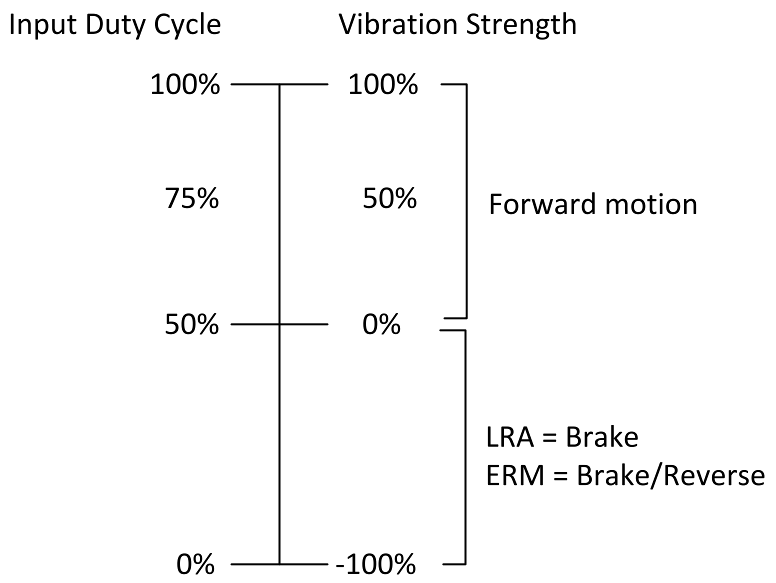


Figure 5. DRV2603's Relationship Between Input PWM Duty Cycle and Vibration Strength

The MSP430TCH5E Haptics Library is designed to interface with the DRV2603 and DRV8601. The *effect banks* it contains are tuned to these devices. In turn, these devices work with a wide array of ERM and LRA actuators, listed in Section 4.2.

2.5 Control of the PWM Duty Cycle: What is a Haptic Effect?

The MSP430TCH5E Haptics Library enables a rich of haptics *effects*, and *sequences* of those effects. (For a more complete description of haptics effects and sequences, see Section 2.7.)

An effect is essentially a mapping of a changing PWM duty cycle over a brief period of time, producing a particular vibration pattern experienced by the user. The library uses one of the TCH5E's Timer_A modules, and one of its output pins, to generate the PWM.

The lifetime of a haptic effect is as follows:

1. The MSP430TCH5E application software calls a library API function that loads the effect into the haptics engine.
2. The library sets the enable signal for the driver device.
3. The library loads an initial duty cycle (sometimes referred to as a voltage) into the Timer_A chosen to output the PWM.

4. Every 5 ms, the application calls on the library to update the duty cycle.
5. Eventually, the effect reaches the end of its mapping of duty cycle over time and thus completes.
6. The library then disables the enable signal for the driver device.

This creates a voltage that changes on a stepwise basis over time, which might implement ramps, curves, or other waveforms.

2.6 Effect Banks: Tuning Duty-Cycle Waveforms to Actuator Types

Even within the ERM or LRA types, actuators vary in their specifications. Obviously there are variances in physical size and power efficiency, but these do not have bearing on how the PWM duty cycle should be controlled.

In contrast, some actuator parameters do impact the duty cycles that should be applied. For example, some actuators have different voltage ratings, moving the line between normal forward drive and overdrive.

Another set of parameters are the response times – for example, the rise time and brake time. The rise time is the time required for the actuator to go from a quiescent state to generating full acceleration, when voltage is applied. The voltage may be the actuator's rated voltage, or it may be an overdrive voltage. The brake time is the time required for the actuator go from the rated full acceleration to a level of acceleration below human perception, while applying the negative overdrive (braking) voltage.

To account for these variances, the MSP430TCH5E Haptics Library includes several *effect banks*. The developer must choose the best effect bank for the chosen actuator, and this requires knowing these parameters.

2.7 Two Types of Haptics: Effects and Sequences

The most basic building block in the MSP430TCH5E Haptics Library is called an *effect.* A haptics effect consists of a pattern of varied voltages applied to the actuator.

For example, one effect found in Appendix A is "Transition Ramp Up Long Sharp". It consists of an initial low voltage (and thus weak haptics vibration) ramping upward to a high voltage (and thus strong haptics effect). Reciprocally, a "Transition Ramp Down Long Sharp" would move from high voltage to low. The library contains 123 of these effects.

The developer can then chain these effects together into *effect sequences.* A sequence consists of one or more effects played sequentially with a programmable time gap between each one; and then the sequence might be repeated one or more times. The variations that are possible using effects and sequences provide tremendous potential for creativity, allowing the developer to target haptics for any given application. Sequences can be created that mimic rolling dice, machine guns, or explosions.

In comparison to sequences, one can think of effects as more simple, linear, and short, in helping to fulfill their role as building blocks for sequences.

The best way to become familiar with effects and sequences is to experience them, using the HapTouch BoosterPack.

2.8 MSP430TCH5E: Integrated-Function and Serial Slave Configurations

In MSP430TCH5E-based systems, the MSP430TCH5E always originates the PWM that drives the actuator. However, the developer has a choice of where the haptics command originates.

- Integrated-Function: Haptics commands originate within the TCH5E software application itself
- Fixed-Function: The TCH5E receives and implements haptics commands from a host MCU

The MSP430TCH5E is equipped with an Universal Serial Communications Interface (USCI) module, which is capable of I^2C, SPI, or UART communication. Any embedded processor – the *host MCU* – can send commands to the TCH5E over these interfaces (see Figure 6).

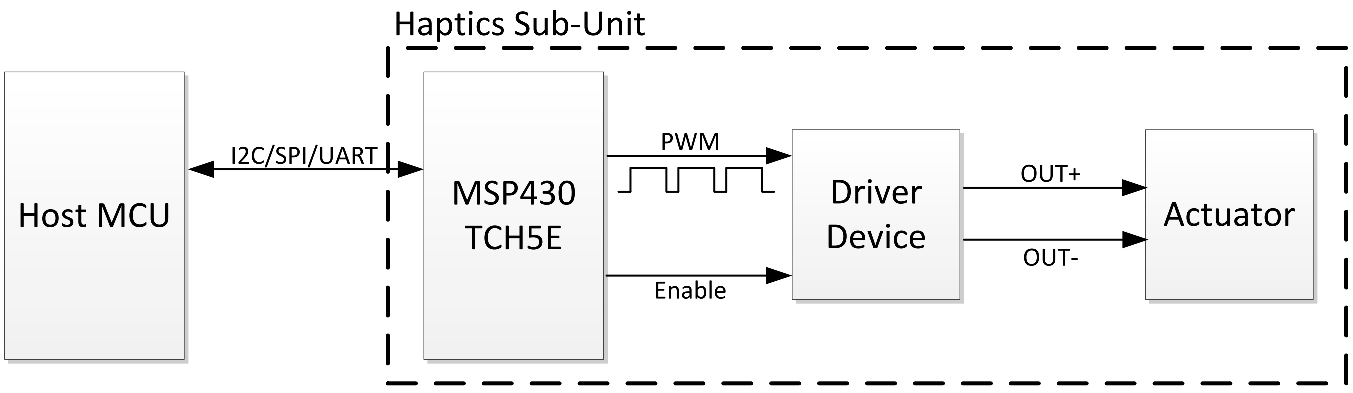


Figure 6. MSP430TCH5E Fixed-Function Serial Slave Configuration

A detailed software example of this is provided with this library, which uses I^{2}C. The example contains a complete serial command set, described in Section 6. The example could be easily ported to SPI or another interface.

Alternatively, because the developer has complete control of the TCH5E application software, and because the TCH5E has plenty of memory beyond what the haptics library requires, software on the TCH5E can be written that receives system inputs and autonomously drives haptics effects based on those inputs, using the library's API. For example, the TCH5E can sample capacitive touch buttons and drive haptics in response (see Figure 7).

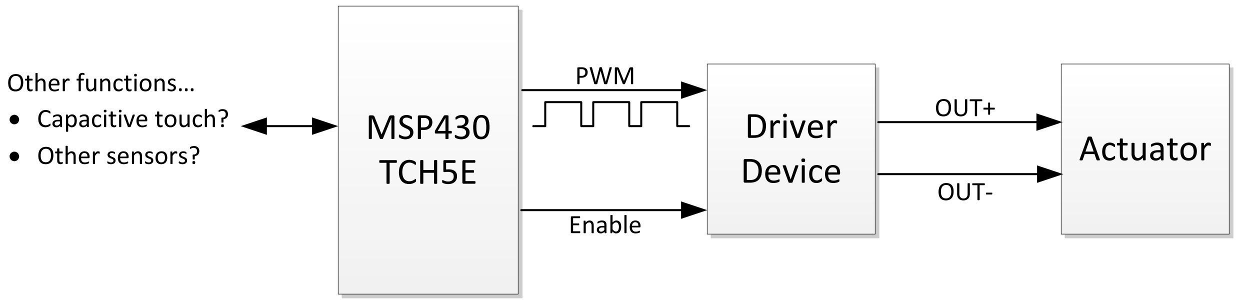


Figure 7. MSP430TCH5E Integrated-Function Configuration

Software examples are provided for both cases. The guide contains a detailed reference for the library's API calls and also contains a reference for the serial interface commands used in the serial slave example.

3 Library Organization and Requirements

The library includes:

- API calls for all haptics functions. (A reference for these API calls is provided in Section 5.)
- Example code for driving haptics effects
- An example serial interface for sending haptics commands from a host MCU. Both MSP430TCH5E and host MCU code is provided. Section 5 serves as a reference guide for this interface.
- Hardware abstraction, in that the developer has a choice of timers and pin allocation for generating the PWM output, as well as flexibility in how to input an audio signal for audio haptics.
- A wide range of ERM and LRA actuators is supported.
- The HapTouch GUI helps the developer in choosing effects and building sequences.

3.1 Hardware Assumptions

The library is provided as part of a turnkey solution. One reason for this is that every component in the system impacts haptic performance, and the effect banks provided in the library have been optimized for the entire system.

3.1.1 MSP430 Device Derivative Requirement

The MSP430TCH5E Haptics Library **only** runs on the MSP430TCH5E device. This device has been specially optimized for haptics. Attempting to run the library on other MSP430 device derivatives will fail, and the API function calls will return an error.

The library does not come pre-programmed into the MSP430TCH5E; rather, it must be downloaded and programmed into each TCH5E device.

3.1.2 Driver Device Requirement

The library is designed to be used in haptics systems where the DRV2603 or DRV8601 is used as the driver device between the MSP430TCH5E and the actuator. This is because the driver devices influence the mapping between duty cycles and the drive voltages applied to the actuator.

3.1.3 Actuator Requirements

The requirements for actuators used with the library are described in Section 4.2. A wide range of ERM and LRA actuators is supported.

3.2 Software Contents

The MSP430TCH5E Haptics Library software is included within the HapTouch BoosterPack software (MSP430-HAPTOUCH-SDK). Its contents are as shown in Table 1.

Table 1. MSP430TCH5E Haptics Library Contents

Item	Description
\MSP430_Haptics_Library	To add the library to your project, add this directory into it, and follow the other instructions in Section 3.7.
\MSP430_Haptics_Library	
HapticsLib.lib	A static library file containing the entire MSP430TCH5E Haptics Library
HapticsLib.h	A header file allowing an application to access the library
License.txt	The MSP430 Haptics Library is bound by a special license as part of its relationship with Immersion Corporation. The license text is found here.
\Examples	
HapticsLibExample_AudioHaptics	An example showing how to use the audio-to-haptics function.
HapticsLibExample_Haptics_from_TCH5E_app	An example showing how to drive haptics from within the MSP430TCH5E application software.
HapticsLibExample_SerialSlave	An example showing how to drive the MSP430TCH5E+library as a serial slave. (The serial interface protocol and commands are described in Section 6.)
I2C_Master_G2553	This is code for the host MCU that sends haptics commands to the MSP430TCH5E slave. It runs on an MSP430G2553 but could be ported to any MCU.
I2C_Slave_TCH5E	Software enabling the TCH5E to run as a fixed-function haptics serial slave device.
Designers_Guide.url	Double-clicking invokes your default browser to download this designer's guide.

The HapticsLib.lib file is provided as object code. All the examples are provided as C source code, contained within complete software projects.

The examples shown above are very simple and minimal. This reduces operations down to their fundamental requirements, and avoids the confusion that can arise from more complex examples.

However, a more complex example is also available: the software for the HapTouch BoosterPack, which is located in the other major directory of the HapTouch BoosterPack software. Most notably, the BoosterPack software combines the haptics library with the MSP430 Capacitive Touch Library. It also shows use both as an integrated-function device and fixed-function serial slave.

3.3 Tool Requirements

The library supports development with Code Composer Studio™ (CCS) and currently does not support development with IAR Embedded Workbench™ or MSPGCC. Because the MSP430TCH5E's flash memory is 16KB, and because the free version of CCS is code-size limited to 16KB, this free version can be used for any TCH5E-based application.

The version of CCS used for development of the library was v5.4. It is expected to work with later versions.

The library is a static library built using the EABI (ELF) output format, and any application using the library must do the same. (The other CCS output format, COFF, has been deprecated.)

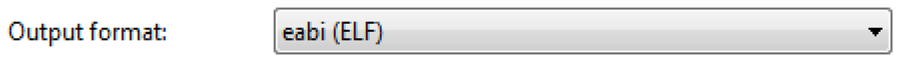

Output format: eabi (ELF)

Figure 8. Output Format in CCS

3.4 Memory Requirements

The library requires a certain amount of flash and RAM memory. Table 2 shows values for the library only, not including the application.

NOTE: These values are only shown as an example; refer to the *.map file in the \Debug directory of the project for an actual breakdown.

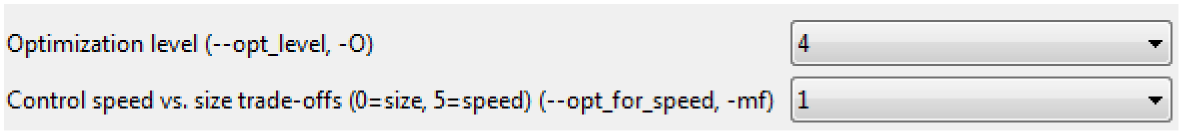
Table 2. Memory Use

Type	Use (bytes)
RAM (not including stack)	163
Flash (code)	2.6K
Each Haptics Library effect bank added with HAPTIC_addBank()	1.8K

The RAM use is entirely static; no heap space is required.

These numbers reflect the library found in v1.00.00.00 of the HapTouch BoosterPack SDK.

The HapticsLib.lib file is built using the optimization settings shown in Figure 9. It uses the highest level of optimization, with an emphasis on code size.


Optimization level (--opt_level, -O) 4

Control speed vs. size trade-offs (0=size, 5=speed) (--opt_for_speed, -mf) 1

Figure 9. HapticsLib.lib Optimization Settings (CCS)

3.5 Use of MCU Peripheral Resources

The library does not directly use any of the TCH5E's peripherals. However, it uses callback functions to access a means of driving a PWM signal, which is usually done with a Timer_A module. The developer can choose which of the two Timer_A modules should be used, and which output pin for that timer will output the signal.

The library also needs certain function calls to be made on certain periods. For example, non-audio haptics applications must call an update function every 5 ms. Or, if using the audio haptics feature, the application must make calls on 1 ms and 250 µs periods. Calling according to these periods usually requires a timer. Although it is possible to use the WDT (watchdog timer) for this purpose, Timer_A has more flexibility in setting its period, and thus the other Timer_A on the TCH5E is often a better choice.

Finally, the audio haptics function requires a means of acquiring a digitized audio signal. Usually this is done with the TCH5E's ADC10 module.

Although these are the typical configurations, they are not required to be this way. For example, if desired, the update calls could be driven from I/O toggles driven from an external source capable of the necessary timing, saving one of the internal timers for other uses.

3.6 Data Types

All integer types used in the example project are C99-compatible.

Table 3. Data Types

Type	Description
`uint8_t`	unsigned 8-bit
`uint16_t`	unsigned 16-bit
`int8_t`	signed 8-bit
`int16_t`	signed 16-bit

The HapticsLib.h file declares a special enumerated type for TRUE and FALSE, which is occasionally used by the library. (In the event that these type names somehow conflict with application-specific defines by the same name, the library will recognize any non-zero value as TRUE, and any zero value as FALSE.)

```
typedef enum
{
    TRUE = 1,
    FALSE = 0
}Boolean;
```

The library also defines some struct types. These are described in this document where their related concepts are described.

3.7 Adding the Library into Your Project

Adding the library to any CCS project consists of the following simple steps.

1. Copy the \MSP430_Haptics_Library directory, containing HapticsLib.lib/h, into the target project
2. #include HapticsLib.h in any source file accessing the library
3. Ensure that include paths are implemented that allow any file calling the library to find HapticsLib.h. This might be done using a global include path in the project settings.

4 Design Considerations

This section describes, in detail, various topics required to implementing haptics using the library and the MSP430TCH5E.

4.1 *Interfacing to an Actuator*

Section 2 gave a general description of the actuator interface, which is also what you need to implement in your system.

The MSP430TCH5E must output two signals to the DRV2603 or DRV8601:

- A PWM
- An enable signal

The TCH5E has two Timer_A instances, each of which is capable of driving this PWM. You must write the callback functions to initialize this PWM and set its duty cycle; see Section 4.3.

Optionally, instead of outputting the enable signal from the MSP430TCH5E, you could permanently enable the driver device. However, this can have a significant impact on the driver's power consumption, and so it is recommended to only enable it when the PWM is being driven.

4.2 *Actuator Selection and Corresponding Effect Banks*

The library supports a wide variety of ERM and LRA actuators. Some have been qualified by Immersion for both performance and durability, and are recommended. However, there are many other good actuators on the market. If you want to use an actuator that is not on Immersion's certified list, see Section 4.2.3.

Either way, it is important that the correct effect bank be chosen.

4.2.1 Effect Banks

In choosing the actuator, an appropriate effect bank must be selected for that actuator. Failure to choose the right bank may result in sub-optimal performance, or could even damage the actuator. See Section 2.6 for additional description of effect banks.

Once chosen, the library needs to be initialized to use this bank, using the `HAPTIC_addBank()` and `HAPTIC_init()` calls.

The Immersion Corporation has defined and created banks as part of the TS2200C TouchSense library, on which the MSP430TCH5E Haptics Library is based. The bank definitions are shown in Table 4.

Table 4. Immersion Effect Bank Definitions

Effect Bank	Actuator Type	Characteristics	Actuator Rated Voltage	Actuator Maximum Voltage	90% Rise Time at 3 V (ms)	90% Brake Time at -3 V (ms)
A	ERM	Best – with overdrive	1.3	3[(1)]	40-60	10-20
B		Very good – fast for light devices	3	3	40-60	5-15
C		Good – strong	3	3	60-80	10-20
D		Acceptable	3	3	100-140	15-25
E		Marginal	3	3	>140	>30
F	LRA	Good	2-V RMS drive (AC 175 Hz)		<30	Approx. 30

(1) Overdrive voltage applied for less than 60 ms at a time.

Section 2.4 describes the actuator parameters in the table.

4.2.2 Using an Immersion-Certified Actuator

Immersion maintains a list of actuators certified for compatibility with the TS2200 TouchSense library, and determines the appropriate effect banks. Certification indicates two things:

- The actuator has passed the TouchSense Actuator Performance test, which validates an actuator's dynamic capabilities for reproducing vibration effects in the TouchSense System. It includes testing for dynamic performance, acceleration, frequency response, and power efficiency.
- The actuator has passed the TouchSense Actuator Life Test, which is a crucial step to achieving durability criteria established by one or more handset manufacturers. This test contains 1,000,000 touch cycles of on/off scenarios.

Certified actuators are shown in Table 5 for each effect bank.

Table 5. Immersion-Certified Actuators

Effect Bank	Make and Model
Bank A	• AWA GT-4168 • ZLIFE RP1342
Bank B	• ZLIFE RF2323
Bank C	• LNLON Y0408A-400050572-M021 • KOTL/Jinlong • Z4TH5B1241993 • DMEGC DM-YK421-7G • DMEGC DM-YK407-6F2 • KOTL/Jinlong Z4TL2B124167S
Bank D	• DMEGC DM-YX403 • DMEGC DM-YX402-A
Bank E	• AWA GH-4342 (bar) • Jinlong C1030B028F (coin)
Bank F	• SEMCO 1036 LRA 175-Hz resonance

4.2.3 Using a Non-Certified Actuator: Procedure for Determining the Appropriate Bank

If you want to use a non-certified actuator, the procedure below details how to determine the appropriate bank. Note that these tests involve bypassing the driver device and applying the full rated voltage, or its inverse braking voltage. All but the first step apply to ERM actuators.

This procedure was part of producing the list in the previous section.

1. If the actuator is LRA, with a definition similar to the one shown in Table 4, select effect Bank F.
2. If the actuator is rated for 1.3 V operation and can be overdriven with 3 V for up to 60 ms at a time, select effect Bank A. (All other banks are designed for a 3-V voltage rating.)
3. If the actuator's 90% rise time at 3 V and 90% brake time at −3 V can be determined from the data sheet, select the effect bank using the bank definitions in Table 4.
4. If these parameters cannot be discerned from the data sheet, you can obtain them by measurement, and then select the effect bank using the bank definitions in Table 4.
   To measure the rise time:
   (a) Attach the actuator to a mass equivalent to the mass of the physical unit to be vibrated with the haptics effect.
   (b) Attach an accelerometer to the mass and view the accelerometer output on an oscilloscope.
   (c) Rest the mass on a compliant surface, such as silicone or soft rubber.
   (d) Apply a step-function +3-V dc voltage to the actuator.
   (e) Using the accelerometer's output, note how long it takes for the acceleration amplitude to reach 90% of the steady-state (maximum) value.

To measure the brake time:

(a) Use the same setup as for the rise time.

(b) Apply a positive pulse as for the rise time allowing the acceleration to reach steady-state.

(c) Step down to −3 V and note the time it takes for the acceleration to fall to 10% of the steady-state value. Do not apply this brake voltage long enough to cause the motor to spin in the reverse direction. To be sure about this, begin with short brake pulses and increase their duration until the exact brake time is determined.

5. If the effect bank cannot be determined by the above steps, then use a subjective process to select the appropriate bank. Try driving some effects using each effect bank, and determine which one feels best. While doing this, ensure that brake pulses driven by the test effects never cause the actuator to spin backward. In other words:

   (a) Attach the actuator to a mass equivalent to the mass of the physical unit to be vibrated with the haptics effect.

   (b) Select an effect bank

   (c) Play a few different effects and note how they feel on the target mass. Use the same set of effects in each effect bank for comparison.

   (d) Play effects #0-5 using the effect bank. These effects are chosen because they use braking. If the actuator ever moves backward while playing or stopping an effect, then the braking pulse was applied too long; select a different effect bank.

   (e) Repeat steps a through d for each effect bank.

   (f) After trying all the effect banks, select the effect bank that seemed to provide the best effects, without the actuator moving in the reverse direction.

Spinning an actuator backward can damage the actuator. It is important to ensure that brake pulses from playing or stopping effects in the selected effect bank never cause the actuator to spin backward. If effects in the selected effect bank cause the motor to spin backward, the effect bank is inappropriate for the actuator and a different effect bank should be selected.

4.2.4 Representation of the Banks Within the Library

The library declares these items related to effect banks.

```
typedef enum
{
    EFFBANK_MOTOR_A = 0,
    EFFBANK_MOTOR_B,
    EFFBANK_MOTOR_C,
    EFFBANK_MOTOR_D,
    EFFBANK_MOTOR_E,
    EFFBANK_MOTOR_F
}EFFECT_BANK;

typedef struct
{
    const uint8_t *peffectPtr;
    EFFECT_BANK nBank;
}DSE_Table;

extern DSE_Table MOTOR_A;
extern DSE_Table MOTOR_B;
extern DSE_Table MOTOR_C;
extern DSE_Table MOTOR_D;
extern DSE_Table MOTOR_E;
extern DSE_Table MOTOR_F;
```

The enumerated type EFFECT_BANK creates an index for each bank, which can then be passed into `HAPTIC_init()` to select the bank the library will use. The names EFFBANK_MOTOR_A, EFFBANK_MOTOR_B, and so on refer to the Bank A, Bank B, and so on described in the previous sections.

The type DSE_Table links each EFFECT_BANK index with an address pointer to the actual bank data within the library; and in the declarations above, external references of this type to each bank are declared. These are used when calling `HAPTIC_addBank()`.

```
HAPTIC_addBank(MOTOR_A);
HAPTIC_init(EFFBANK_MOTOR_A, enableActCB, disableActCB, initActCB, setActDriveLevelCB);
```

Only banks that have been added with `HAPTIC_addBank()` can be referenced in `HAPTIC_init()`.

Calling `HAPTIC_addBank()` is what causes that particular bank to be linked into the final application image. Banks not added are not linked in. This is significant because each bank requires between 1.5K and 2K of flash. Therefore, to add banks not actually used would waste flash memory.

If more than one actuator is physically attached to the MSP430TCH5E, then it is possible to add a bank for each actuator, and dynamically call `HAPTIC_init()` to select whichever one is active. A separate set of callback functions would then also be needed (see Section 4.3).

The selection made with `HAPTIC_init()` applies to both non-audio and audio applications.

4.3 Hardware Flexibility Through the Callback Functions

To enable the developer flexibility in choosing pins, peripherals, and layout, callbacks serve as an abstraction layer between the library and the hardware.

Table 6. Defined Callback Functions

Callback	Required Content
`enableActCB()`	Enables the actuator; starts the PWM, and enables the line driver
`disableActCB()`	Disables the actuator; stops the PWM, and disables the line driver
`initActCB()`	Initializes the haptics line driver device
`setActDriveLevelCB(uint8_t)`	Set the PWM duty cycle
`sampleAdcCB()`	(Audio only) Obtains a sample from the audio source

As a result of these callbacks, the following flexibility is provided:

- Either Timer_A can be used to drive the PWM
- Any output pin on that timer can be used;
- Any I/O can be used to drive the enable signal (or none at all, if the driver's enable pin is statically asserted)
- Audio sources other than ADC10 could be used

The developer must write code for these callbacks. Pointers to the callbacks are then passed into the library with the `HAPTIC_init()` function. If audio haptics is to be implemented, the `HAPTIC_audioInit()` function must also be called, which sets the audio-specific callback function.

The callbacks defined in the examples are probably very close to what is needed for most applications.

4.4 Initializing the Library

If not using the audio feature, initialization is fairly simple.

First, determine the appropriate effect bank, as described in Section 4.2. Then, ensure that the callbacks are properly defined, as described in the previous sections.

Then call:

```
HAPTIC_addBank(MOTOR_A);
HAPTIC_init(EFFBANK_MOTOR_A, enableActCB, disableActCB, initActCB, setActDriveLevelCB);
```

The library is now ready for the application to make API calls to drive haptic effects.

If the audio feature is to be used, then two additional calls must be made:

```
HAPTIC_audioInit(sampleAdcCB);

HAPTIC_audioConfigStr audioSettings;
audioSettings.midpoint = 128;
audioSettings.wakeupThresh = 10;
audioSettings.inputMin = 20;
audioSettings.inputMax = 64;
audioSettings.strength = 127;
audioSettings.strengthAtFloor = 0;

HAPTIC_audioConfig(&audioSettings);
```

The settings for `HAPTIC_audioConfig()`, as well as the `HAPTIC_audioConfigStr`, are described in Section 4.8.

4.5 Calling the Update Functions

The haptics engine must continuously be pushed by periodic update calls. The calls to be made, and when, are shown in Table 7.

Table 7. Haptic Engine Update Calls

Function	When
Non-Audio Applications	
`HAPTIC_update()`	Every 5 ms ± 10%
Audio Applications	
`HAPTIC_getConversion()`	Every 250 µs ± 10%
`HAPTIC_audioUpdate()`	Every 1 ms ± 10%

Without making these calls, the engine essentially goes to sleep.

In non-audio applications, after an effect or sequence is loaded into the queue with `HAPTIC_playEffect()` or `HAPTIC_playSequence()`, it must be fully played out (or stopped with `HAPTIC_stopEngine()`) before another one can be loaded. Until then, calling `HAPTIC_isEngIdle()` will return FALSE, because it is waiting for the loaded effect or sequence to be played. `HAPTIC_update()` must be called every 5 ms until the loaded effect or sequence has completed.

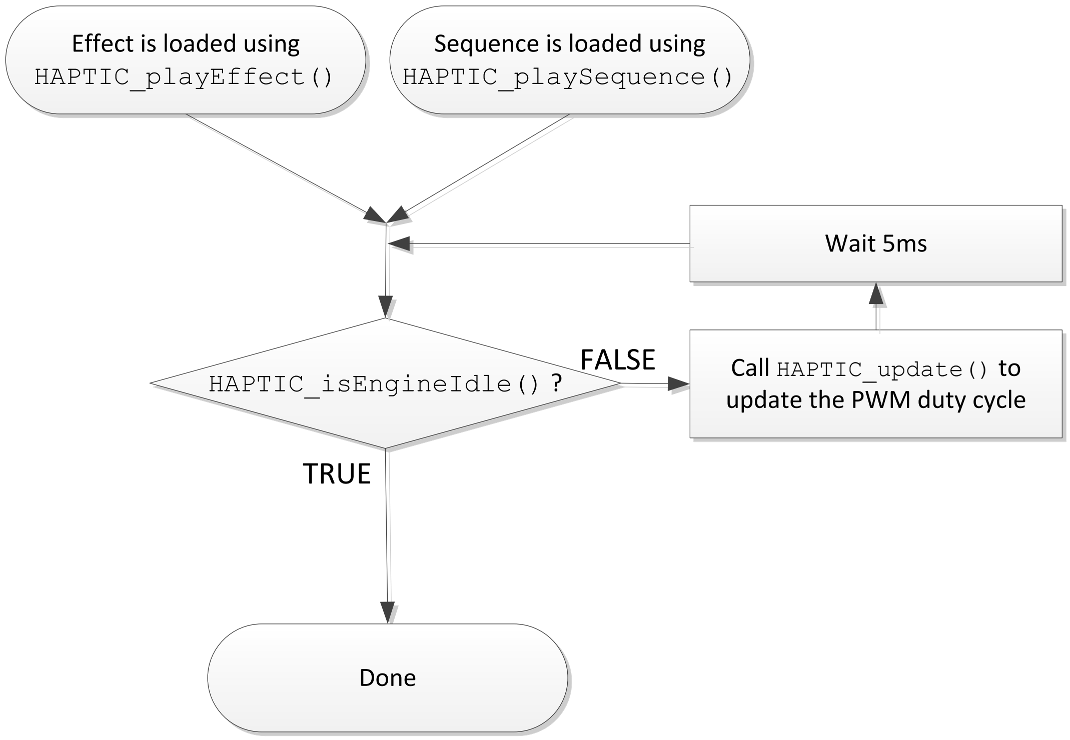

Effect is loaded using HAPTIC_playEffect()
Sequence is loaded using HAPTIC_playSequence()
Wait 5ms
HAPTIC_isEngineIdle() ?
FALSE
Call HAPTIC_update() to update the PWM duty cycle
TRUE
Done

Figure 10. Haptics Engine Cycle for Playing a Haptics Effect or Sequence

In audio applications, `HAPTIC_getConversion()` first calls the `sampleAdcCB()` callback to sample the audio signal, and then processes it. Every four times that `HAPTIC_getConversion()` is called, `HAPTIC_audioUpdate()` must also be called, and this function updates the voltage applied to the actuator based on the audio samples that have been received since the last call. (See Section 4.8 for more information about the audio haptics engine.)

4.6 Haptics Effects

As described earlier, the *effect* is the fundamental building block of the haptics generated by the library. The library contains 123 such effects (bearing in mind that two of these are null effects). The library contains a constant in HapticsLib.h that defines how many there are:

```
#define TOTAL_EFF_COUNT 123
```

Most effects last under one second, and are fairly simple in nature. Most are fairly homogeneous or linear. This is consistent with their role as elemental building blocks.

The function call `HAPTIC_playEffect()` can be used to play individual effects. As a parameter, it accepts an effect index `effIndex` to be played, according to the list of effects given in Appendix A. Valid indices are between zero and `TOTAL_EFF_COUNT - 1`.

4.7 Haptics Sequences

Effects can then be chained together into *sequences.* A sequence consists of several *effect pairs,* with each pair consisting of an `effIndex` and a `timeGap`.

The `timeGap` is a delay that will be placed after the corresponding effect, during which no haptics will be generated. Because of the 5-ms period of `HAPTIC_update()`, the minimum time unit by which sequences are played is a 5-ms tick. The time gap is expressed in the number of 5-ms ticks. It can be anywhere from 0 ms to 1.27 seconds.

Finally, the sequence has a `repeatCount` – the number of times the sequence is to be repeated in succession.

Figure 11 describes what happens when a sequence is played. In relation to Figure 10, it can be thought of as containing the next level of detail. As indicated in Figure 10, if at any point in this flowchart `HAPTIC_isEngineIdle()` is called, it will return FALSE.

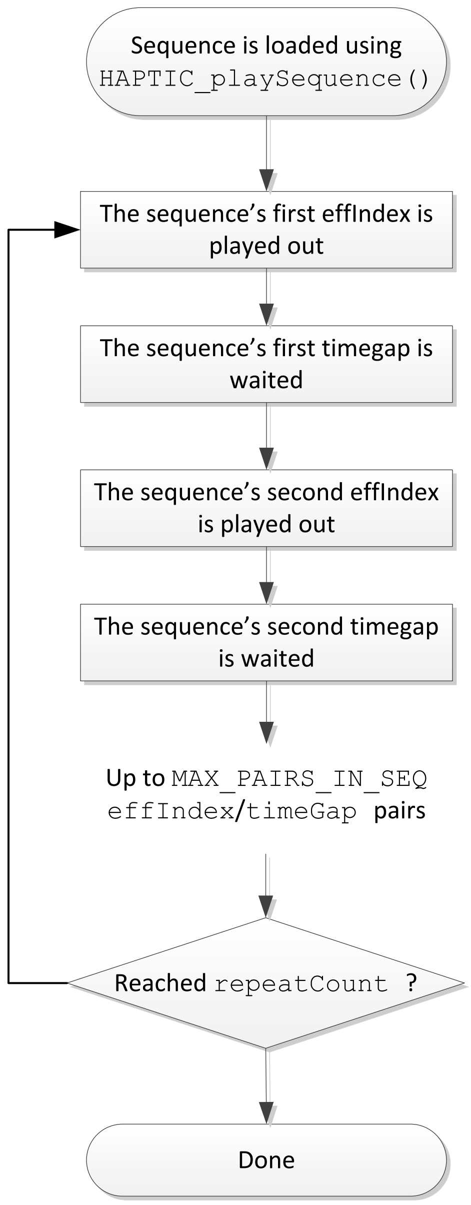

Sequence is loaded using HAPTIC_playSequence()
The sequence's first effIndex is played out
The sequence's first timegap is waited
The sequence's second effIndex is played out
The sequence's second timegap is waited
Up to MAX_PAIRS_IN_SEQ effIndex/timeGap pairs
Reached repeatCount ?
Done

Figure 11. Chronological Playback of a Sequence

In software, a sequence is defined by an instance of HAPTIC_sequence*:*

```
#define MAX_PAIRS_IN_SEQ     16
#define MY_MAX_PAIRS_IN_SEQ  5   // application-defined

typedef struct
{
    uint8_t effIndex; // from zero to TOTAL_EFF_COUNT - 1
    uint8_t timeGap;  // counted in 5-ms ticks
}HAPTIC_seqEffPair;

typedef struct
{
    uint8_t pairCount;    // the number of valid pairs in effpairs[] for this instance
    uint8_t repeatCount;  // the number of times the sequence should be repeated
    HAPTIC_seqEffPair effPairs[MY_MAX_PAIRS_IN_SEQ];
}HAPTIC_sequence;
```

A pointer to an instance of this struct is passed into the function `HAPTIC_playSequence()`.

`MAX_PAIRS_IN_SEQ` is a value fixed by the library. `MY_MAX_PAIRS_IN_SEQ` is a value you can configure; the only requirement is that it not exceed `MAX_PAIRS_IN_SEQ`. The reason it exists is that if every instance of `HAPTIC_sequence` allocates RAM for the number of pairs defined by `MAX_PAIRS_IN_SEQ`, this can add up to a significant amount of RAM. `MY_MAX_PAIRS_IN_SEQ` gives you the opportunity to reduce the amount of RAM allocated per instance of `HAPTIC_sequence`.

4.8 Audio Haptics Engine

The audio haptics engine converts a stream of audio data into haptic effects. Audio data is usually provided to the library from the MSP430TCH5E's on-chip ADC10 module, although alternate sources could be used. The library then generates haptics effects.

The software for initializing and maintaining this functionality looks a little different than it does for non-audio applications. It is described in this section, but you may also wish to see the example included with this library called `HapticsLibExample_AudioHaptics`; see Table 1.

The amplitude of the incoming audio stream is analyzed. Large amplitudes are converted into strong haptics effects; small ones into slight effects. An example where this feature might be used is in a game controller playing a video game; a loud explosion would cause the actuator to shake the controller, helping the person playing the game to experience the explosion.

The application must call `HAPTIC_getConversion()` every 250 µs. This means it is called at a 4kHz rate. One of the first things this function does is call the `sampleAdcCB()` callback, which the developer must write, to somehow obtain a sample of the audio signal. Usually this is done with the ADC10 module. The library expects an 8-bit response, and so the two least significant bits of the ADC10 output would need to be discarded. `HAPTIC_getConversion()` then performs analysis of the signal.

Then, on 1 ms intervals, the application must also call `HAPTIC_audioUpdate()`. (Note that this is one call for every four calls of `HAPTIC_getConversion()`.) This function further processes the data, determines a new voltage at which the actuator should be driven, and applies it to the actuator using the callbacks.

Given the 4-kHz sampling rate, it is expected that hardware pre-filtering is applied, limiting the incoming signal to under 2 kHz to avoid aliasing.

During initialization, the `AUDIOHAPTIC_init()` function should be called, after `HAPTIC_init()`. This should be followed by `AUDIOHAPTIC_config()` function, through which a number of parameters are provided that instruct the library how to convert the audio into voltages for the actuator. These parameters are shown in Table 8.

Table 8. Audio-to-Haptics Parameters

Parameter	Range	Description
`midpoint`	0-255	The value of the audio signal stream when there is no audio. If an incoming audio signal has a bias voltage applied, this is the location within the ADC's numeric 8-bit output range at which the bias voltage resides.
`wakeupThresh`	0-255[1]	The threshold value, wherein if the signal amplitude stays underneath it, the function `HAPTIC_audioNoSignal()` will return TRUE. This should be chosen as a value safely above the analog circuitry's noise floor. The value is expressed as units relative to the midpoint.
`inputMin`	0-255[1]	A signal with an amplitude at this value will cause effects to be driven of the strength indicated by `strengthAtFloor`. The value is expressed as units relative to the midpoint, and should be less than `inputMax`.
`inputMax`	0-255[1]	As signal amplitude increases from `inputMin` toward `inputMax`, the haptics effects will increase linearly. Once at `inputMax`, the corresponding haptics effects will be of the strength indicated by `maxStrength`. Amplitudes above `inputMax` will still generate at `maxStrength`. The value is expressed as units relative to the midpoint, and should be greater than `inputMin`.
`strengthAtFloor`	0-255[1]	The strength at which the actuator will be driven when the amplitude is at `inputMin`. The value should be less than `maxStrength`. The value is expressed as units relative to the midpoint, and should be less than `maxStrength`.
`maxStrength`	0-255[1]	The strength at which the actuator will be driven when the amplitude is at `inputMax`. The value is expressed as units relative to the midpoint, and should be less than `strengthAtFloor`.

[1] Although the maximum value is shown as 255 in each case, it can only be this large if the midpoint is zero. The actual maximum limit is (255 - `midpoint`).

An example waveform is shown in Figure 12. In the figure, `midpoint` has been set to half of the 256-value range, or 128. All the other values are expressed as relative to that value, and it is this relative scale that is shown at left.

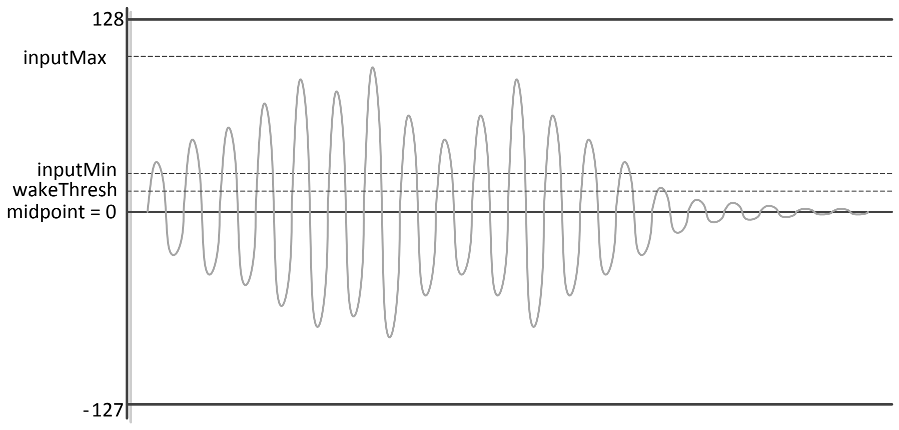


Figure 12. Example Audio Config Values With Midpoint Set Halfway Through Range (128)

Figure 13 shows example settings when the `midpoint` has been set much lower, at 30. Notice that the scale at left is different, because all the values are shown relative to the midpoint.

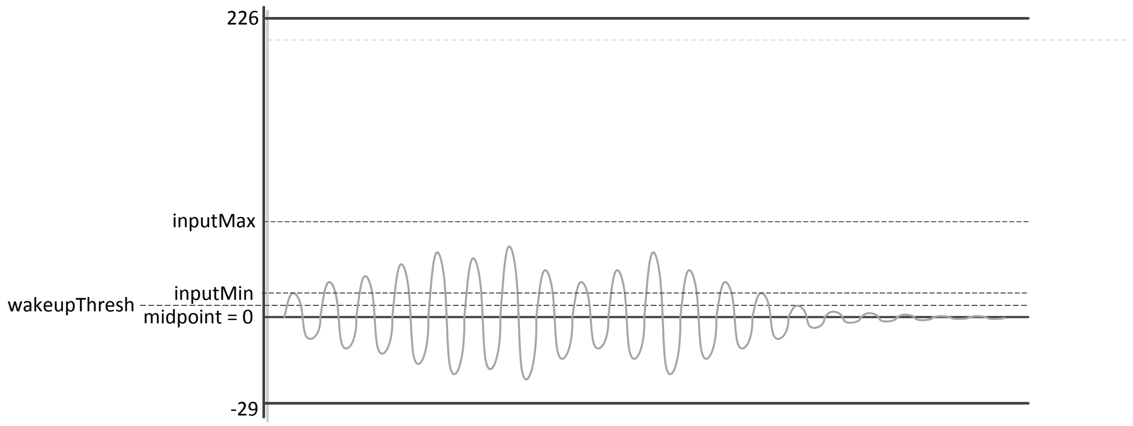


Figure 13. Example Audio Config Values With Midpoint Set at 30

If using ADC10, the choice of reference voltage is up to you. Whether 1.5 V or 2.5 V is chosen, the same 10-bit range is produced, and so it does not matter. This is primarily determined by the design of the analog interface.

4.9 Serial Interface

As described in Section 3.2, the example HapticsLibExample_SerialSlave shows how to use the TCH5E as a serial slave. Example code is shown for both the MSP430TCH5E itself, as well as for the host MCU. Although just an example and specific to I^{2}C, it might be usable as-is for some applications. For this reason, the serial interface has its own reference section in this document (see Section 6).

The serial interface provides access to all the same functions as if called from the MSP430TCH5E application; but being a serial interface, it does not provide all the same flexibility. For example, whether the commands originate within the MSP430TCH5E software or from a host MCU, the TCH5E software is still responsible for calling the periodic update functions and configuring the timers and ADC10. These must be set at compile-time, not handled through the serial interface.

4.10 Power Management

In non-audio applications, while the haptics engine is playing back an effect or sequence that has been loaded, the application is free to enter a low-power mode, as long as it maintains the PWM continuously until the effect or sequence has been fully played. Once it has been fully played, the library requires nothing of the MCU (CPU or clocks) while the non-audio haptics engine is idle.

Audio applications, in contrast, involve continuous activation of the PWM output. This continuous operation is driven by repeated calls to `HAPTIC_audioGetConversion()` and `HAPTIC_audioUpdate()`. Ceasing to call these essentially puts the engine to sleep.

The decision to sleep during audio haptics might be made by system-level factors, but a common case might be to put the engine to sleep when there is no audio signal. The library tracks signal activity against the `wakeupThresh` value that was set by `HAPTIC_audioConfig()`. The application can then call `HAPTIC_audioIsNoSignal()`, which returns a boolean indicating whether there has been a signal recently. If not, the application might choose to significantly slow the calls to `HAPTIC_audioGetConversion()`. If a later call to `HAPTIC_audioIsNoSignal()` says a signal has returned, the rate of calling `HAPTIC_audioGetConversion()` can return to 4 kHz, and calling of `HAPTIC_audioUpdate()` can resume.

4.11 Clock and Timer Management

If the application does not use the audio feature, then there are generally no restrictions on MCLK speed, provided that the PWM frequency is properly managed, as described in Section 2.3.

As mentioned previously, in non-audio applications, `HAPTIC_update()` must be called every 5 ms ± 10%. This completes fairly quickly, and so it can be called out of the interrupt service routine (ISR) of a timer. The next questions to be answered are, which timer, and using which clock?

The timing is easy to accomplish with a Timer_A module driven by a calibrated DCO or a 32-kHz crystal. However, the VLO's tolerance is too wide to accomplish ±10% timing. One option is to run the timer from the DCO while `HAPTIC_isEngIdle()` returns FALSE, but switch to the VLO otherwise (or disable the timer altogether), allowing the MSP430 to rest in LPM3 or LPM4.

The timing is a little difficult to accomplish with the watchdog timer (WDT) using the standard calibrated DCO frequencies (1, 8, 12, or 16 MHz), because its clock selection is not as flexible. However, it is possible to do.

If using the audio feature, `HAPTIC_audioGetConversion()` must be called every 250 µs. Every fourth time, `HAPTIC_audioUpdate()` must also be called. This means both functions must execute within a 250-µs window, and it can be difficult to do this with an MCLK speed below 15 MHz.

Both of these audio functions can be executed out of a Timer_A ISR. Using the TI-calibrated 16-MHz DCO speed for MCLK is always safe for audio; but if using a slower speed, it is recommended to measure the execution of these functions and ensure they are completing within the 250-µs window.

5 Haptics Library API Calls

This section is a reference guide for the API calls.

5.1 Return Codes

Most of the API functions return a single-byte return code of type `uint8_t`. Table 9 shows the possible values.

Table 9. Possible Return Codes

Pointer	Description
`HAPTIC_SUCCESS`	The operation was a success.
`HAPTIC_ENG_NOT_IDLE`	The haptics-playing operation could not be completed, because the engine is currently playing another effect or sequence; and the function was instructed not to override effects underway.
`HAPTIC_QUEUE_FULL`	The haptics-playing operation could not be completed, because the queue was full. This means the loaded effect or sequence has not yet begun to be played, and often means that `HAPTIC_update()` is not being called as it should be.
`HAPTIC_UNSUPPORTED_DEV`	All functions in the library check to ensure the MSP430 derivative being used is the MSP430TCH5E. If not, they fail, sometimes in unclear ways. The only function that returns a code unique to this kind of failure (that is, `HAPTIC_UNSUPPORTED_DEV`) is `HAPTIC_init()`.

5.2 Callbacks

Four callback functions must be defined, both in non-audio and audio applications. These are communicated to the library in `HAPTIC_init()`. If using the audio feature, `sampleAdcCB()` must also be defined, and this is communicated to the library in `HAPTIC_audioInit()`. See Section 4.3 for description of the callback functions.

Table 10. Haptics Library Callbacks

Name	User-Implemented Functionality
`initActCB()`	Perform any initialization functions related to the actuator, including preparing the PWM output.
`enableActCB()`	Electrically enable the actuator.
`disableActCB()`	Electrically disable the actuator
`setActDriveLevelCB(uint8_t)`	Adjust the duty cycle of the PWM driving the actuator
`sampleAdcCB()`	Cause a sampling of the audio signal to occur

5.2.1 initActCB()

Description

Called by the library to perform any actions necessary to initialize the haptics actuator. This consists of:

- Configure the chosen Timer_A's output frequency
- Set a 50% duty cycle (which equates to no haptic motion)
- Prepare the PWM output pin, and initialize its output to low

The TCH5E has two Timer_A instances, and one of these must be chosen to drive the actuator's PWM. Any output pin for that Timer_A instance can be chosen, giving flexibility to arrange timers and pins within the application, as needed.

See Section 2.3 for a description of how to choose the PWM frequency.

The developer can choose to activate the timer in this function and only enable the output in `enableActCB()`, or to not enable the timer in this function, and enable both the timer and output in `enableActCB()`. Either way is acceptable, with no particular benefit either way.

For information about configuring Timer_A as a PWM, see the *MSP430x2xx Family User's Guide* (SLAU144), the examples accompanying this library, and the MSP430TCH5E code examples.

Parameters

	passed in/out	None
Returns		None

Example

```
void initActuator(void)
{
    P3DIR |= (MOTOR_EN_PIN | MOTOR_PWM_PIN);  // Set PWM/enable pins as outputs
    P3OUT &= ~(MOTOR_EN_PIN | MOTOR_PWM_PIN); // Drive out low (driver EN is active-high)
    P3SEL |= MOTOR_PWM_PIN;                   // Select function 'timer output'
    TA1CCR0 = 255;                            // PWM period. (Be aware of changes to SMCLK!)
    TA1CCR1 = 127;                            // CCR1. Sets the PWM duty cycle
    TA1CTL = TASSEL_2 + MC_1;                 // Use SMCLK as input clock; UP mode
    TA1CCTL1 &= ~OUTMOD_7;                    // Fixes the output to low
}
```

5.2.2 enableActCB()

Description

This is called by the library when haptics effects are to be driven. Its job is to enable the driver device, and begin driving the PWM signal out of the TCH5E.

Parameters

	passed in/out	None
Returns		None

Example

```
void enableActuator(void)
{
    P3OUT |= MOTOR_EN_PIN; // Set the driver enable to high, enabling the actuator driver
    TA1CCTL1 |= OUTMOD_7;  // Sets the output to reset/set mode, activating the timer output
}
```

5.2.3 disableActCB()

Description

This is called by the library when haptics effects are completed. Its job is to de-activate the PWM signal and the driver device.

Parameters

	passed in/out	None
Returns		None

Example

```
void disableActuator(void)
{
    P3OUT &= ~MOTOR_EN_PIN; // Set the driver enable to low, disabling the actuator driver
    TA1CCTL1 &= ~OUTMOD_7;  // Fixes the output to low, de-activating the timer output
}
```

5.2.4 setActDriveLevelCB()

Description

The library calls this every 5 ms throughout the playing of an effect (within `HAPTIC_update()`) to change the duty cycle of the PWM driven by the chosen Timer_A instance and output pin.

The value dutyCycle is a number from 0 to 255, where 0 means 0% duty cycle and 255 means 100%. The value should be converted into a value relative to the CCR0 value that set the PWM period in `initActCB()`, such that the result divided by the CCR0 value is the same as `dutyCycle`/255. This result should then be assigned to the Timer_A CCRx register corresponding with the PWM output pin.

Parameters

`uint8_t dutyCycle`	passed in	A number from 0 to 255, where 0 = 0% duty cycle and 255 = 100%
	passed out	None
Returns		None

Example

```
void setActuatorDriveLevel(uint8_t dutyCycle)
{
    TA1CCR1 = dutyCycle;
}
```

5.2.5 sampleAdcCB()

Description

This function is called by the library early within a call to `HAPTIC_audioGetConversion()`. Because the latter function is called every 250 µs, this callback is also called on that same period. Note that this produces an audio sampling rate of 4 kHz.

The callback should return a sampling of the input audio signal, which typically comes from the on-chip ADC10 module.

The value must be 8 bit and in the range from 0 to 255. Because ADC10 outputs 10-bit values, this generally requires throwing away the two least significant bits.

Parameters

	passed in/out	None
`uint8_t`	Returned	An 8-bit audio value

Example

```
uint8_t sampleADC(void)
{
    ADC10CTL0 |= (ENC | ADC10SC);  // Sampling and conversion start
    while (ADC10CTL1 & ADC10BUSY); // Wait for the conversion to finish
    return ADC10MEM >> 2;          // ADC10 outputs 10-bit, but the library requires 8-bit
}
```

5.3 Configuration Calls

These are calls that configure the haptics library for operation, necessary for both non-audio and audio haptics.

Generally speaking, calls specific to audio haptics have the word "audio" as the first word after the "`HAPTIC_`" prefix.

Table 11. Haptics Library Callbacks

Function	Description
`HAPTIC_addBank()`	Registers the bank for potential use by the library
`HAPTIC_init()`	Initializes the library, selects a bank, and assigns the callback functions to be used
`HAPTIC_audioInit()`	Initializes the audio haptics function, and assigns the audio haptics callback function
`HAPTIC_audioConfig()`	Configures the audio haptics input signal parameters

5.3.1 HAPTIC_addBank()

Description

Registers an effect bank for use by the library. The primary effect of doing so is that this particular bank becomes linked into the application's object code. Each bank consumes between 1.5K and 2K. The application should only add banks needed by the attached actuators.

Only banks that have been added can be referenced in `HAPTIC_init()`. Because most applications only have one actuator, it is usually the case that only one bank needs to be added.

Parameters

`DSE_Table actuatorTable`	passed in	One of these predefined DSE_Table options, corresponding with the available effect banks: MOTOR_A MOTOR_B MOTOR_C MOTOR_D MOTOR_E MOTOR_F
	passed out	None
`uint8_t result`	returned	HAPTIC_SUCCESS

Example

```
HAPTIC_addBank(MOTOR_E);
```

5.3.2 HAPTIC_init()

Description

Initializes the effect sequence engine, and prepares the effect bank. This function needs to be called after adding the banks.

The library only runs on an MSP430TCH5E device. Although all the library functions will fail, or operate unreliably, if they are executed on a device other than the MSP430TCH5E, this particular function is unique in that it will return HAPTIC_UNSUPPORTED_DEV if executed in this way.

Parameters

`EFFECT_BANK bankIndex`	passed in	The index of the chosen bank. Must be one that was previous added with `HAPTIC_addBank()`.
`void* enableActCB`	passed in	A pointer to the callback function for enabling the actuator
`void* disableActCB`	passed in	A pointer to the callback function for disabling the actuator
`void* initActCB`	passed in	A pointer to the callback function for initializing the actuator
`void* setActDriveLevelCB`	passed in	A pointer to the callback function for setting the actuator's PWM duty cycle
	passed out	None
`uint8_t result`	returned	HAPTIC_SUCCESS HAPTIC_UNSUPPORTED_DEV

Example

```
HAPTIC_init(bankIndex,
            enableActCB,
            disableActCB,
            initActCB,
            setActDriveLevelCB);
```

5.3.3 HAPTIC_audioInit()

Description

Initializes the audio haptics engine. This must be called first, before a later call to `HAPTIC_audioUpdate()`. If the audio haptics engine is to be used, this function must follow any call to `HAPTIC_init()`.

`HAPTIC_addBank()` and `HAPTIC_init()` must be called prior to `HAPTIC_audioInit()`.

Parameters

`void*`	passed in	A pointer to the callback function for sampling the audio signal
	passed out	None
`uint8_t result`	returned	HAPTIC_SUCCESS

Example

```
HAPTIC_audioInit(sampleAdcCB);
```

5.3.4 HAPTIC_audioConfig()

Description

Initializes the parameters by which the audio haptics engine converts the input audio signal into haptics effects. These parameters are described in Section 4.8.

This function must be called after `HAPTIC_audioInit()`, for proper operation. The default values are undefined.

Parameters

HAPTIC_audioConfigStr* settings	passed in	A pointer to a HAPTIC_audioConfigStr that has been populated with the appropriate values.
	passed out	None
uint8_t result	returned	HAPTIC_SUCCESS

Example

```
HAPTIC_audioConfig(settings);
```

5.4 Standard Haptics API Calls

These are calls used for non-audio haptics applications.

See Section 4.5 through Section 4.7 for a description of the life cycles of a haptics effect and sequence, which is what these functions govern or of which they return status.

Table 12. Haptics Library Callbacks

Function	Description
`HAPTIC_playEffect()`	Load an effect into the haptics engine's queue. It will be played, as `HAPTIC_update()` is subsequently and repeatedly called.
`HAPTIC_playSequence()`	Load a sequence of effects into the haptics engine's queue. It will be played, as `HAPTIC_update()` is subsequently and repeatedly called.
`HAPTIC_stopEngine()`	Ends playback of any effects or sequences currently underway
`HAPTIC_update()`	Performs the next incremental step in playing any effects or sequences in the engine's queue. Must be called every 5 ms.
`HAPTIC_isEngIdle()`	Indicates whether an effect or sequence is currently being played.

5.4.1 HAPTIC_playEffect()

Description

Loads a new effect into the haptics engine's queue, and returns. As `HAPTIC_update()` is called every 5 ms, the effect will be played.

If `timeout` is non-zero, then the effect will be stopped after the indicated period has passed.

If `bOverride` is TRUE, then any effects or sequences currently being played will be aborted, and the new effect played. If FALSE, then if an effect or sequence is already underway, it will be allowed to complete, and this function will return HAPTIC_ENG_NOT_IDLE.

Parameters

`uint8_t effIndex`	passed in	The effect to be played; between zero and (`TOTAL_EFF_COUNT` - 1)
`uint16_t timeout`	passed in	The effect will stop playing after timeout × 5 ms. If zero, then no timeout will be applied.
`uint8_t bOverride`	passed in	Boolean indicating how the function should respond if an effect is already underway. TRUE means it stops the previous effect or sequence and then plays the new one. FALSE means it will return with the code HAPTIC_ENG_NOT_IDLE.
	passed out	None
`uint8_t result`	returned	HAPTIC_SUCCESS HAPTIC_ENG_NOT_IDLE HAPTIC_QUEUE_FULL

Example

```
HAPTIC_playEffect(1, 0, 0, FALSE);
```

5.4.2 HAPTIC_playSequence()

Description

Loads a new sequence into the haptics engine's queue, and returns. As `HAPTIC_update()` is called every 5 ms, the effect will be played.

The sequence will be repeated `repeatCount` times. If `repeatCount` is zero, it will not be repeated; meaning it will only be played once.

If `bOverride` is TRUE, then any effects or sequences currently being played will be aborted, and the new effect played. If FALSE, then if an effect or sequence is already underway, it will be allowed to complete, and this function will return HAPTIC_ENG_NOT_IDLE.

Parameters

`HAPTIC_sequence* seq`	passed in	A pointer to a properly populated `HAPTIC_sequence` structure.
`uint8_t bOverride`	passed in	Boolean indicating how the function should respond if an effect is already underway. TRUE means it stops the previous effect or sequence and then plays the new one. FALSE means it will return with the code HAPTIC_ENG_NOT_IDLE.
	passed out	None
`uint8_t result`	returned	HAPTIC_SUCCESS HAPTIC_ENG_NOT_IDLE HAPTIC_QUEUE_FULL

Example

```
HAPTIC_sequence seq;
seq.pairCount = 3;
seq.repeatCount = 2;
seq.effPairs[0].effIndex = 4;
seq.effPairs[0].timeGap = 2;
seq.effPairs[1].effIndex = 5;
seq.effPairs[1].timeGap = 2;
seq.effPairs[2].effIndex = 6;
seq.effPairs[2].timeGap = 2;
HAPTIC_playSequence(&seq, TRUE);
```

5.4.3 HAPTIC_stopEngine()

Description

Stops any sequence currently playing, and clears it from the haptics engine's effect queue.

This function does not apply to the audio haptic engine.

Parameters

	passed in	None
	passed out	None
`uint8_t result`	returned	HAPTIC_SUCCESS

Example

```
HAPTIC_stopEngine();
```

5.4.4 HAPTIC_update()

Description

Causes the library to make the next adjustment in actuator PWM value according to the haptics effect or sequence being played. It does this using the user-defined callback functions. Must be called every 5 ms.

Parameters

	passed in	None
	passed out	None
`uint8_t result`	returned	HAPTIC_SUCCESS

Example

```
HAPTIC_update();
```

5.4.5 HAPTIC_isEngineIdle()

Description

Returns FALSE if an effect or sequence is currently being played by the haptics engine; returns TRUE if the engine is idle.

Parameters

	passed in	None
	passed out	None
`boolean`	returned	TRUE: engine is idle FALSE: engine is not idle, and instead is in the process of playing a haptics effect or sequence

Example

```
If(!HAPTIC_isEngineIdle())
{
    HAPTIC_playSequence(&seq, FALSE);
}
```

5.5 Audio Haptics API Calls

These calls are specific to the audio haptics feature.

Table 13. Haptics Library Callbacks

Function	Description
`HAPTIC_audioGetConversion`	Prompts the library to get the next sample of audio data and analyze it
`HAPTIC_audioUpdate`	Prompts the library to update the PWM value, according to the result of processing the last several audio samples
`HAPTIC_audioNoSignal`	Indicates whether the incoming audio signal's amplitude is above or below the wakeupThresh value submitted by `HAPTIC_audioConfig()`.
`HAPTIC_audioShutdown`	Puts the actuator into an idle position, and prepares the library for a period of `HAPTIC_audioGetConversion()` and `HAPTIC_audioUpdate()` not being called on their usual periods.

5.5.1 HAPTIC_audioGetConversion()

Description

Prompts the library to get a sample of the incoming audio signal and analyze it. It samples the audio using the `sampleAdcCB()` callback. The application should call this function every 250 µs.

Parameters

	passed in	None
	passed out	None
`uint8_t result`	returned	HAPTIC_SUCCESS

Example

```
HAPTIC_audioGetConversion();
```

5.5.2 HAPTIC_audioUpdate()

Description

Prompts the library to update the PWM value applied to the actuator, based on recent audio samples obtained with `HAPTIC_audioGetConversion()`. This function should be called every 1 ms, following `HAPTIC_audioGetConversion()`.

Parameters

	passed in	None
	passed out	None
`uint8_t result`	returned	HAPTIC_SUCCESS

Example

```
AUDIOHAPTIC_update();
```

5.5.3 HAPTIC_audioIsNoSignal()

Description

Returns whether the library has identified a recent audio signal above the wakeupThresh value that was configured using `HAPTIC_audioConfig()`.

The function counts the number of times out of the last 16 calls to `HAPTIC_audioGetConversion()` that the audio signal's peak amplitude remains under `wakeupThresh`. If the count reaches 16, `HAPTIC_audioIsNoSignal()` begins to return TRUE. As the samples begin to be above `wakeupThresh`, the count begins to decrement.

As such, it takes 16 successive low-amplitude samples to be counted as a lack of audio. But if the amplitude hovers near `wakeupThresh`, the count will remain high, and the response of this function may fluctuate.

This function can be used as an indication of whether the period of calling `HAPTIC_audioGetConversion()` can be slowed, as well as the calling of `HAPTIC_audioUpdate()`.

Parameters

	passed in	None
	passed out	None
`boolean result`	returned	TRUE: No audio present FALSE: Audio present

Example

```
uint8_t nowActive = !HAPTIC_audioNoSignal();

// If transition from audio to no audio
if(wasActive && !nowActive)
{
    HAPTIC_audioShutdown();
    TA0CCR0 = TIMERA_DURING_NOAUDIO_SIGNAL;
}

// If transition from no audio to active audio
if(!wasActive && nowActive)
{
    TA0CCR0 = TIMERA_DURING_AUDIO_SIGNAL;
}

if(nowActive)
{
    HAPTIC_audioUpdate();
}

wasActive = nowActive;
```

5.5.4 HAPTIC_audioShutdown()

Description

Stops the continuous driving of the actuator during audio haptics, and prepares the library for a period of slowed activity.

Parameters

	passed in	None
	passed out	None
`uint8_t result`	returned	HAPTIC_SUCCESS

Example

```
if(HAPTIC_audioNoSignal())
{
    HAPTIC_audioShutdown();
    slowTheTimer();
}
```

6 I^{2}C Serial Interface Commands

The example HapticsLibExample_SerialSlave, located within the SDK zip file, shows use of the MSP430TCH5E and library as an I^{2}C serial slave device. The serial commands used in that interface are described in this section.

The example actually contains two pieces of code: one for the MSP430TCH5E slave, and one for an I^{2}C master interfacing to it. In the example, an MSP430G2553 is used as the example slave; as a result, it can be executed on a HapTouch BoosterPack mounted into a G2 LaunchPad. See the application report *Haptics and Capacitive Touch Using the MSP430TCH5E: the HapTouch BoosterPack* (SLAA616).

6.1 I^{2}C Command Transaction Format

All the I^{2}C transactions in this protocol consist of the host MCU writing a command to the TCH5E, perhaps with some parameters. Then, after a brief delay, the response is read back from the TCH5E.

In I^{2}C, each slave has an address, and only responds to reads or writes to that address. The master must use this address to get a response from the slave. This value is defined in both the master and slave examples:

```
#define I2C_SLAVE_ADDR_OF_TCH5E 0x10
```

In all transactions, the first byte written in the transaction toward the TCH5E is the command byte. Each command byte corresponds with a command detailed in Section 6.2.

Most commands are then followed by parameter bytes. Once sent, the write portion of the transaction is terminated with a STOP condition.

Table 14. Command Write (Host → TCH5E)

Byte Offset	Field	Direction	Description
		Host → TCH5E	I^{2}C START condition
		Host → TCH5E	Slave address + W
0	Command byte	Host → TCH5E	
1	Parameter #0	Host → TCH5E	
2	Parameter #1	Host → TCH5E	
⋮	⋮	⋮	
		Host → TCH5E	I^{2}C STOP condition

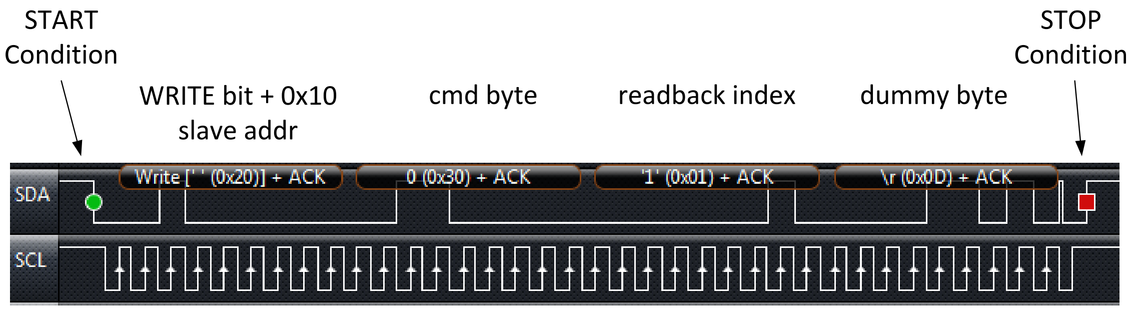


Figure 14. Host MCU Sending I^{2}C Command to MSP430TCH5E Slave

The command in Figure 14, 0x30, is a CMD_READBACK. CMD_READBACK is always followed by a readback index; in this case, 0x01, which is RB_PING. The dummy byte is not part of the protocol, but rather resolves a problem that can occur with the USCI module when multiple system interrupts are present that cause period timing delays, which can cause the I^{2}C STOP ISR to be handled prior to the last byte being received.

After sending the command, the host MCU reads the response. In I²C reads, the host chooses how many bytes to read within the boundaries of a START and STOP condition. In this protocol, the host MCU knows how many bytes to expect for each command. Values are provided in the code, in the form of CMD_LEN_xxxx, where xxxx reflects the command that was called.

Most commands return a single byte in their response, a *command response* byte. The exception is CMD_READBACK, which may or may not return a command response, and always returns other data as well, depending on what the readback index was.

Table 15. Response Read (Except for CMD_READBACK)

Byte Offset	Field	Direction	Description
		Host → TCH5E	I²C START condition
		Host → TCH5E	Slave address + R
0	Response code	Host ← TCH5E	
		Host → TCH5E	I²C STOP condition

Table 16. Response Read (CMD_READBACK)

Byte Offset	Field	Direction	Description
		Host → TCH5E	I²C START condition
		Host → TCH5E	Slave address + R
0	Parameter #0	Host ← TCH5E	
1	Parameter #1	Host ← TCH5E	
⋮	⋮	⋮	
		Host → TCH5E	I²C STOP condition

Figure 15 shows the response to the command that was sent in Figure 14.

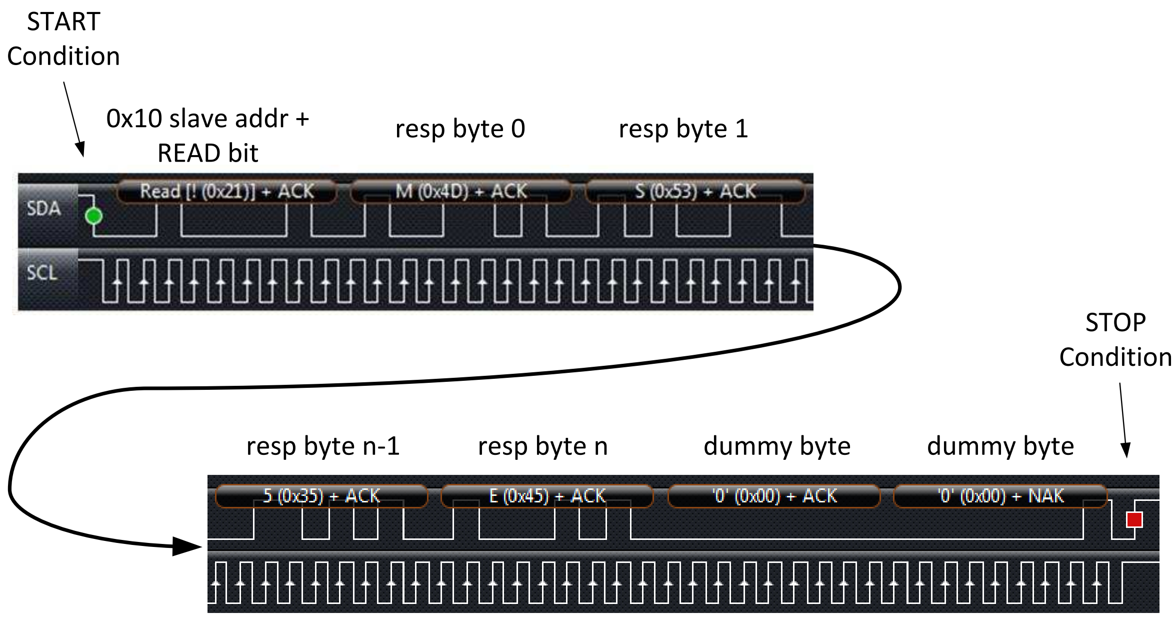


Figure 15. Host MCU Reading Response From the MSP430TCH5E Slave

It will take the MSP430TCH5E software some time to complete the command. Until it does, the reads will return the value `CMDRESP_WAIT_FOR_RESP`.

6.2 Command Detail

The serial commands used in the example are listed in Table 17, as well as the haptics library function calls that result from the serial commands.

Table 17. Haptics Library Callbacks

Serial Command	Description
Non-Audio	
CMD_PLAY_EFFECT	Calls `HAPTIC_playEffect()`
CMD_PLAY_SEQ	Calls `HAPTIC_playSequence()`
CMD_STOP_PLAYBACK	Calls `HAPTIC_stopEngine()`
CMD_READBACK	Reports status information about the haptics functionality
CMD_RESET	Causes the TCH5E to perform a PUC (power-up clear) reset on itself
Audio-Related	
CMD_AUDIOHAPTICS_ENABLE	Calls `HAPTIC_audioEnable()`
CMD_AUDIOHAPTICS_CONFIG	Calls `HAPTIC_audioConfig()`

The command CMD_READBACK can be used to pull any type of data from the TCH5E. In the current implementation, is only has one valid index. However, more could easily be added.

Unlike other commands, the readback response does not need to have a response code as its first byte.

Table 18. Readback Indices

Serial Command	Description
RB_PING	Returns the string "MSP430TCH5E". Can be used to determine if the TCH5E device is alive and receiving commands.

For any command other than CMD_READBACK, the only byte returned during the read phase is a response code. The commands all share a common set of codes; they are detailed below.

Most of these responses are generated by code that validates the command format, and if anything is incorrect, the command is rejected before attempted. The exceptions are CMDRESP_QUEUE_FULL and CMDRESP_ENG_NOT_IDLE, which generally means the slave attempted to act on the command, often by calling a haptics library API. The response code in these cases often reflects what the API call returned.

Table 19. Serial Command Response Codes

Status Byte	Description
`CMDRESP_SUCCESS`	The command succeeded.
`CMDRESP_INVALID_CMD`	The command value was invalid.
`CMDRESP_BAD_PARAM_LEN`	There were too many parameter bytes.
`CMDRESP_QUEUE_FULL`	The effects queue was full, and as a result no effect (or sequence) was loaded.
`CMDRESP_TIME_OOR`	The timeout value passed in with the command was not between 0 and 2000.
`CMDRESP_EFF_INDEX_OOR`	The effect index was not between 0 and TOTAL_EFF_COUNT - 1.
`CMDRESP_INVAL_RB_INDEX`	The readback index was invalid.
`CMDRESP_REPEAT_OOR`	The repeat count was out of range.
`CMDRESP_INVALID_INPUT`	The input was invalid, in some way not specified by the codes above.
`CMDRESP_ENG_NOT_IDLE`	An attempt was made to play an effect or sequence, but the haptics engine was not idle and the 'override' parameter had been set to false; therefore the attempt aborted.
`CMDRESP_WAIT_FOR_RESP`	Until the MSP430TCH5E has finished handling the command, this is the value the host MCU receives on the I^{2}C interface.
`CMDRESP_CMD_INVALID_DURING_AUDIO`	During audio haptics, the software is configured to respond with this response code for any command other than one that causes it to exit audio haptics.

6.2.1 CMD_PLAY_EFFECT

Description

This command calls the function `HAPTIC_playEffect()`. See the reference for that command, for more information about what this command does.

Parameters

Byte Offset	Field	Description
0	Command	The command byte: 0x54
1	effect	The index of the effect to be played. Must be between zero and TOTAL_EFF_COUNT - 1.
2	timeout (low byte)	If non-zero, imposes a timeout that stops the effect after timeout × 5 ms. The total value must be ≤2000d. If zero, no timeout is imposed.
3	timeout (high byte)	
5	bOverride	zero: if the effect engine is currently playing an effect or sequence when this command is made, the function will return CMDRESP_ENG_NOT_IDLE and abort. non-zero: if the effect engine is currently playing an effect or sequence when this command is made, the previous effect or sequence will be aborted, and the effect sent here will be played.

Return

Byte Offset	Field	Description
0	Response codes	CMDRESP_SUCCESS CMDRESP_QUEUE_FULL CMDRESP_EFF_INDEX_OOR CMDRESP_TIME_OOR CMDRESP_BAD_PARAM_LEN CMDRESP_ENG_NOT_IDLE

6.2.2 CMD_PLAY_SEQ

Description

This command calls the function `HAPTIC_playSequence()`. See the reference for that command, for more information about what this command does.

Note that the input parameter list is roughly correlated with the standard definition for an effect sequence. The only difference is that there is no `pairCount` value; instead, the list of effect pairs is terminated with a 0xFF signature. The effect sequence is preceded by the command byte, and followed by the repeat count. Everything in between is as specified in Section 4.7.

Parameters

Byte Offset	Field	Description
0	Command ID	0x50
1	Effect index #0	Must be between zero and TOTAL_EFF_COUNT - 1.
2	Gap #0	The number of 5-ms ticks the haptics engine should wait before generating the next effect. Must be 0-254.
3	Effect index #1	Must be between zero and TOTAL_EFF_COUNT - 1.
4	Gap #1	The number of 5-ms ticks the haptics engine should wait before generating the next effect. Must be 0-254.
⋮	⋮	
⋮	Last effect index	
n-3	Last effect gap	
n-2	End signature	0xFF
n-1	Repeat Count	The number of times the sequence will be repeated. If zero, then it will not be repeated, meaning that it plays only once. Must be 0-254.
n	Override	zero: if the effect engine is currently playing an effect or sequence when this command is made, the function will return `CMDRESP_ENG_NOT_IDLE` and abort. non-zero: if the effect engine is currently playing an effect or sequence when this command is made, the previous effect or sequence will be aborted, and the sequence sent here will be played.

Return

Byte Offset	Field	Description
0	Response codes	CMDRESP_SUCCESS CMDRESP_QUEUE_FULL CMDRESP_EFF_INDEX_OOR CMDRESP_BAD_PARAM_LEN CMDRESP_INVALID_INPUT CMDRESP_TIME_OOR CMDRESP_REPEAT_OUTOFRANGE CMDRESP_ENG_NOT_IDLE

6.2.3 CMD_STOP_PLAYBACK

Description

This command calls the function `HAPTIC_stopEngine()`. See the reference for that command, for more information about what this command does.

Parameters

Byte Offset	Field	Description
0	Command ID	0x52

Return

Byte Offset	Field	Description
0	Response codes	CMDRESP_SUCCESS

6.2.4 CMD_READBACK

Description

This command returns various kinds of information from the haptics library. An index determine which readback table is returned.

The one predefined readback index, RB_PING, can be used as a means of verifying that communication with the MSP430TCH5E is working properly.

Parameters

Byte Offset	Field	Description
0	Command ID	0x30
1	Readback_index	Determines which information is returned by the readback command, according to the following table.

Readback Indices

Readback Table Index	Readback Table
0	RB_PING

Readback Table #0: Ping

Byte Offset	Field	Description
0	Data	"M"
1	Data	"S"
2	Data	"P"
3	Data	"4"
4	Data	"3"
5	Data	"0"
6	Data	"T"
7	Data	"C"
8	Data	"H"
9	Data	"5"
10	Data	"E"

6.2.5 CMD_AUDIOHAPTICS_ENABLE

Description

This command calls the function `HAPTIC_audioEnable()` or `HAPTIC_audioDisable()`, depending on the parameter. See the reference for that command for more information about what this command does.

Parameters

Byte Offset	Field	Description
0	Command ID	0x40
1	Enable_Disable	Non-zero: Enable Zero: Disable

Return

Byte Offset	Field	Description
0	Response codes	CMDRESP_SUCCESS

6.2.6 CMD_AUDIOHAPTICS_CONFIG

Description

This command calls the function `HAPTIC_audioConfig()`. See the reference for that command, for more information about what this command does.

Note that the data format roughly correlates with the definition of a `HAPTIC_audioConfigStr` structure. See Section 4.8 for information on how to configure these parameters.

Parameters

Byte Offset	Field	Description
0	Command ID	0x41
1	midpoint	Must be between 0 and 255
2	wakeupThresh	Must be between 0 and (255 - `midpoint`)
3	inputMin	Must be between 0 and (255 - `midpoint`)
4	inputMax	Must be between 0 and (255 - `midpoint`)
5	strengthAtFloor	Must be between 0 and 255
6	strengthMax	Must be between 0 and 255

Return

Byte Offset	Field	Description
0	Response codes	CMDRESP_SUCCESS

7 Suggested Reading and Resources

7.1 LaunchPad Websites

More information about the G2 LaunchPad, supported BoosterPacks, and available resources can be found at:

- The G2 LaunchPad's tool page: resources specific to the G2 LaunchPad
- TI's LaunchPad portal: information about all LaunchPads from TI, for all MCUs
- The LaunchPad wiki: design resources and example projects from the community

7.2 Information on the MSP430TCH5E

At some point, you will probably want detailed information about the MSP430TCH5E device. For every MSP430 device, the documentation is organized as shown in Table 20.

Table 20. How MSP430 Device Documentation is Organized

Document	For MSP430TCH5E	Description
The user's guide for the device's "family"	MSP430x2xx Family User's Guide (SLAU144)	Architectural information about the device, including clocks, timers, CPU, all peripherals, and other features
The device's data sheet	MSP430TCH5E data sheet (SLAS895)	Device-specific information and all parametric information for this device

7.3 Information on the DRV2603 and DRV8601

You will need information on these driver devices, which are part of the MSP430TCH5E solution. Data sheets can be found in the product folders: DRV2603 and DRV8601. At the time of writing, only a very limited data sheet is obtained when clicking on the data sheet link. The full data sheet is available by clicking on the "Request the data sheet" link below the data sheet link (see Figure 16).

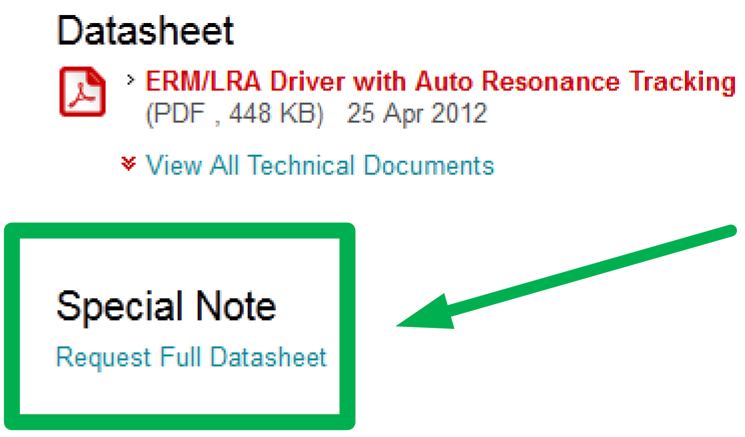

Datasheet
ERM/LRA Driver with Auto Resonance Tracking
(PDF , 448 KB) 25 Apr 2012
View All Technical Documents
Special Note
Request Full Datasheet

Figure 16. DRV2603 Product Page: Requesting the Full Data Sheet

7.4 Download Code Composer Studio

You will need one of these development environments.

As of this writing, the library supports development with only Code Composer Studio. Development with IAR Embedded Workbench and MSP430GCC has not been tested.

7.5 Actuator Information

Working with haptics requires the combination of several disciplines at once: electrical hardware, software, and a unique application of mechanical principles. This library, document, and solution have tried to address the first two, but a complete description of the mechanical aspects is outside the scope of this solution.

The following resources are suggested as a means of learning haptics actuators:

- www.precisionmicrodrives.com: Precision Microdrives is a vendor of haptics actuators. They provide a wealth of information for someone starting out with haptics, addressing in particular the hardware and mechanical aspects. Although not included in Immersion's certified list, their actuators can be adapted using the procedure in Section 4.2.3.
- Other actuator vendors may have good resources as well.
- www.immersion.com: Immersion is the industry leader in haptics, and originated the TouchSense TS2200C library on which the MSP430TCH5E Haptics Library is based. They maintain information there, including links to vendors of certified actuators.
- The documents on the product pages for the DRV2603 and DRV8601 can be useful in learning about actuators. (This includes the data sheets themselves.)

7.6 MSP430Ware and the TI Resource Explorer

MSP430Ware is a complete collection of libraries and tools. By default, MSP430Ware is included in a CCS installation. CCS and MSP430GCC users must download it separately from www.ti.com/tool/msp430ware.

MSP430Ware includes the TI Resource Explorer, for easily browsing the tools.

7.7 MSP430TCH5E Code Examples

This is a set of very simple code examples that demonstrate how to use the MSP430TCH5E's entire set of peripherals (for example, ADC10 and Timer_A). Every MSP430 derivative has a set of these code examples. When writing code that uses a peripheral, they can often serve as a starting point. These examples are available in the MSP430TCH5E product folder.

7.8 MSP430 Application Notes

There are many application notes at www.ti.com/msp430, with practical design examples and topics.

7.9 TI E2E Community

Search the forums at e2e.ti.com. If you cannot find your answer, post your question to the community.

7.10 Community at Large

Many online communities focus on the MSP430 – for example, www.43oh.com. You can find additional tools, resources, and support from these communities.

Appendix A List of Haptics Effects

Table 21 shows all of the individual haptics effects that can be driven from the MSP430TCH5E Haptics Library. The far right columns give suggested scenarios in which the effect might be used in an end application.

To help narrow down your options, see the HapTouch GUI, provided within the same HapTouch SDK zip file that contains this library. The GUI contains filters that help you quickly narrow down to the effect you want.

Table 21. Haptics Effects

Effect Index (in hex)	Description	Button	Alert	Gesture
0	Strong Click – 100%	X		
1	Strong Click – 60%	X		
2	Strong Click – 30%	X		
3	Sharp Click – 100%	X		
4	Sharp Click – 60%	X		
5	Sharp Click – 30%	X		
6	Soft Bump – 100%	X		
7	Soft Bump – 60%	X		
8	Soft Bump – 30%	X		
9	Double Click – 100%	X		
A	Double Click – 60%	X		
B	Triple Click – 100%	X	X	X
C	Soft Fuzz – 60%		X	X
D	Strong Buzz – 100%		X	X
E	Empty	NA	NA	NA
F	Empty	NA	NA	NA
10	Strong Click 1 – 100%	X		
11	Strong Click 2 – 80%	X		
12	Strong Click 3 – 60%	X		
13	Sharp Click 4 – 30%	X		
14	Medium Click 1 – 100%	X		
15	Medium Click 2 – 80%	X		
16	Medium Click 3 – 60%	X		
17	Sharp Tick 1 – 100%	X		
18	Sharp Tick 2 – 80%	X		
19	Sharp Tick 3 – 60%	X		
1A	Short Double Click Strong 1 – 100%	X		
1B	Short Double Click Strong 2 – 80%	X		
1C	Short Double Click Strong 3 – 60%	X		
1D	Short Double Click Strong 4 – 30%	X		
1E	Short Double Click Medium 1 – 100%	X		
1F	Short Double Click Medium 2 – 80%	X		
20	Short Double Click Medium 3 – 60%	X		
21	Short Double Sharp Tick 1 – 100%	X		
22	Short Double Sharp Tick 2 – 80%	X		
23	Short Double Sharp Tick 3 – 60%	X		
24	Long Double Sharp Click Strong 1 – 100%	X		
25	Long Double Sharp Click Strong 2 – 60%	X		

Table 21. Haptics Effects (continued)

Effect Index (in hex)	Description	Button	Alert	Gesture
26	Long Double Sharp Click Strong 3 – 30%	X		
27	Long Double Sharp Click Strong 4 – 100%	X		
28	Long Double Sharp Click Medium 1 – 100%	X		
29	Long Double Sharp Click Medium 2 – 80%	X		
2A	Long Double Sharp Click Medium 3 – 60%	X		
2B	Long Double Sharp Tick 1 – 100%	X		
2C	Long Double Sharp Tick 2 – 80%	X		
2D	Long Double Sharp Tick 3 – 60%	X		
2E	Buzz 1 – 100%		X	X
2F	Buzz 2 – 80%		X	X
30	Buzz 3 – 60%		X	X
31	Buzz 4 – 40%		X	X
32	Buzz 5 – 20%		X	X
33	Pulsing Strong 1 – 100%		X	X
34	Pulsing Strong 2– 60%		X	X
35	Pulsing Medium 1 – 100%		X	X
36	Pulsing Medium 2 – 60%		X	X
37	Pulsing Sharp 1 – 100%		X	X
38	Pulsing Sharp 2– 60%		X	X
39	Transition Click 1 – 100%		X	X
3A	Transition Click 2 – 80%		X	X
3B	Transition Click 3 – 60%		X	X
3C	Transition Click 4 – 40%		X	X
3D	Transition Click 5 – 20%		X	X
3E	Transition Click 6 – 10%		X	X
3F	Transition Hum 1– 100%		X	X
40	Transition Hum 2 – 80%		X	X
41	Transition Hum 3 – 60%		X	X
42	Transition Hum 4 – 40%		X	X
43	Transition Hum 5 – 20%		X	X
44	Transition Hum 6 – 10%		X	X
45	Transition Ramp Down Long Smooth 1 – 100 to 0%		X	X
46	Transition Ramp Down Long Smooth 2 – 100 to 0%		X	X
47	Transition Ramp Down Medium Smooth 1 – 100 to 0%		X	X
48	Transition Ramp Down Medium Smooth 2 – 100 to 0%		X	X
49	Transition Ramp Down Short Smooth 1 – 100 to 0%		X	X
4A	Transition Ramp Down Short Smooth 2 – 100 to 0%		X	X
4B	Transition Ramp Down Long Sharp 1 – 100 to 0%		X	X
4C	Transition Ramp Down Long Sharp 2 – 100 to 0%		X	X
4D	Transition Ramp Down Medium Sharp 1 – 100 to 0%		X	X
4E	Transition Ramp Down Medium Sharp 2 – 100 to 0%		X	X
4F	Transition Ramp Down Short Sharp 1 – 100 to 0%		X	X
50	Transition Ramp Down Short Sharp 2 – 100 to 0%		X	X
51	Transition Ramp Up Long Smooth 1 – 0 to 100%		X	X
52	Transition Ramp Up Long Smooth 2 – 0 to 100%		X	X
53	Transition Ramp Up Medium Smooth 1 – 0 to 100%		X	X
54	Transition Ramp Up Medium Smooth 2 – 0 to 100%		X	X

Table 21. Haptics Effects (continued)

Effect Index (in hex)	Description	Button	Alert	Gesture
55	Transition Ramp Up Short Smooth 1 – 0 to 100%		X	X
56	Transition Ramp Up Short Smooth 2 – 0 to 100%		X	X
57	Transition Ramp Up Long Sharp 1 – 0 to 100%		X	X
58	Transition Ramp Up Long Sharp 2 – 0 to 100%		X	X
59	Transition Ramp Up Medium Sharp 1 – 0 to 100%		X	X
5A	Transition Ramp Up Medium Sharp 2 – 0 to 100%		X	X
5B	Transition Ramp Up Short Sharp 1 – 0 to 100%		X	X
5C	Transition Ramp Up Short Sharp 2 – 0 to 100%		X	X
5D	Transition Ramp Down Long Smooth 1 – 50 to 0%		X	X
5E	Transition Ramp Down Long Smooth 2 – 50 to 0%		X	X
5F	Transition Ramp Down Medium Smooth 1 – 50 to 0%		X	X
60	Transition Ramp Down Medium Smooth 2 – 50 to 0%		X	X
61	Transition Ramp Down Short Smooth 1 – 50 to 0%		X	X
62	Transition Ramp Down Short Smooth 2 – 50 to 0%		X	X
63	Transition Ramp Down Long Sharp 1 – 50 to 0%		X	X
64	Transition Ramp Down Long Sharp 2 – 50 to 0%		X	X
65	Transition Ramp Down Medium Sharp 1 – 50 to 0%		X	X
66	Transition Ramp Down Medium Sharp 2 – 50 to 0%		X	X
67	Transition Ramp Down Short Sharp 1 – 50 to 0%		X	X
68	Transition Ramp Down Short Sharp 2 – 50 to 0%		X	X
69	Transition Ramp Up Long Smooth 1 – 0 to 50%		X	X
6A	Transition Ramp Up Long Smooth 2 – 0 to 50%		X	X
6B	Transition Ramp Up Medium Smooth 1 – 0 to 50%		X	X
6C	Transition Ramp Up Medium Smooth 2 – 0 to 50%		X	X
6D	Transition Ramp Up Short Smooth 1 – 0 to 50%		X	X
6E	Transition Ramp Up Short Smooth 2 – 0 to 50%		X	X
6F	Transition Ramp Up Long Sharp 1 – 0 to 50%		X	X
70	Transition Ramp Up Long Sharp 2 – 0 to 50%		X	X
71	Transition Ramp Up Medium Sharp 1 – 0 to 50%		X	X
72	Transition Ramp Up Medium Sharp 2 – 0 to 50%		X	X
73	Transition Ramp Up Short Sharp 1 – 0 to 50%		X	X
74	Transition Ramp Up Short Sharp 2 – 0 to 50%		X	X
75	Long buzz for programmatic stopping – 100%, 10 seconds max		X	X
76	Smooth Hum 1 (No kick or brake pulse) – 50%		X	X
77	Smooth Hum 2 (No kick or brake pulse) – 40%		X	X
78	Smooth Hum 3 (No kick or brake pulse) – 30%		X	X
79	Smooth Hum 4 (No kick or brake pulse) – 20%		X	X
7A	Smooth Hum 5 (No kick or brake pulse) – 10%		X	X
7B	Empty	NA	NA	NA
7C	Empty	NA	NA	NA
7D	Empty	NA	NA	NA
7E	Empty	NA	NA	NA
7F	Empty	NA	NA	NA

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		